Цель работы

Освоить на практике шифры перестановки.

Выполнение лабораторной работы

Требуется реализовать:

1. Маршрутное шифрование.
2. Шифрование с помощью решеток.
3. Табоица Виженера

Маршрутное шифрование

Текст разбивается на равные блоки N длиной M. Если в конце не хватает букв, то они добавляются в конец. Блоки записываются построчно в таблицу. Затем буквы выписываются по столбцам, которые упорядываются согласно паролю: внизу таблицы приписывается слово из n неповторяющихся букв и столбы нумеруются по алфавитному порядку букв пароля

Чтобы реализовать программу был написан код на python:

1. Функции заполнения матрицы текстом
2. Функция, шифрующая матрицу

```python
import math
import re


1 usage
def is_pass_valid(password, columns):
    if len(password) != columns:
        print("pass len should be equal to row len")
        raise ValueError
    alphabeticValues = re.findall(r'[a-zA-Za-яA-Я]', password)
    if len(alphabeticValues) != len(password):
        print("pass should contain only eng and ru letters")
        raise ValueError


1 usage
def is_len_columns_valid(columns, inputString):
    if columns > len(inputString):
        print("len columns should be <= len of Input String")
        raise ValueError
    if columns <= 1:
        raise ValueError


1 usage
def to_dict(inputString, password, columns, row):
    list_of_slices = []
    for i in range(row):
        list_of_slices.append(inputString[columns * i:columns * (i + 1)])
```

```
    while len(list_of_slices[-1]) != columns:
        list_of_slices[-1] += 'a'

    routeDict = {}

    for i in range(columns):
        stringToAppend = ""
        for j in range(row):
            stringToAppend += list_of_slices[j][i]
        routeDict[password[i]] = stringToAppend

    sorted_dict = dict(sorted(routeDict.items()))

    return sorted_dict


inputedString = input("Input string to encrypt: ")
columns = int(input("Input int value to determine the number columns: "))
password = input("Password: ").lower()

inputString = inputedString.replace(" ", "")  # remove spaces
password = password.replace(" ", "")  # remove spaces
password = ''.join([password[i] for i in range(len(password) - 1) if password[i + 1] != password[i]] + [
    password[-1]])  # remove dublicates
is_len_columns_valid(columns, inputString)
is_pass_valid(password, columns)
row = math.ceil(len(inputString) / int(columns))

routeDict = to_dict(inputString, password, columns, row)
```

```
routeDict = to_dict(inputString, password, columns, row)

cryptogram = ""
for key in routeDict:
    cryptogram += routeDict[key]

print("начальная фраза: ", inputedString)
print("криптограмма: ", cryptogram)
```

Вывод программы (пример как в методических материалах).

```
"C:\Users\Asuser\Desktop\учеба\магистратура\inf sec\lab2\Script
Input string to encrypt:  нельзя недооценивать противника
Input int value to determine the number columns: 6
Password: пароль
начальная фраза:   нельзя недооценивать противника
криптограмма:  еенпнзоатаьовокннеьвлдирияцтиа

Process finished with exit code 0
```

Шифрование с помощью решеток

Строится квадрат из k чисел. Затем к нему добавляются еще 3 квадрата, которые поворачиваются на 90 градусов и получается большой квадрат 2k размерностью. Дальше из большого квадрата вырезаются клетки и прорези записываются буквы. Когда заполнятся все прорези решето поворачивается на 90 градусов. И так продолжается пока не заполнится вся таблица. И буквы выписываются по алфивитному порядку пароля.

Чтобы реализовать программу был написал код на python:

```python
import numpy as np

1 usage
def generate_grid(k):
    grid = np.zeros((k, k))
    for i in range(k):
        for j in range(k):
            grid[i][j] = i * k + j + 1
    return grid

1 usage
def encrypt_with_grid(grid, text):
    k = len(grid)
    encrypted = np.chararray((2*k, 2*k), unicode=True)
    encrypted[:] = ''

    idx = 0
    for _ in range(4):
        for i in range(k):
            for j in range(k):
                if grid[i][j] and idx < len(text):
                    encrypted[i][j] = text[idx]
                    idx += 1

        grid = np.rot90(grid)

    return encrypted

```

```python
1 usage
def extract_by_password(grid, password):
    order = sorted(range(len(password)), key=lambda x: password[x])
    return ''.join([''.join(grid[:, i]) for i in order])

1 usage
def encrypt(text, password):
    k = int(len(text)**0.5)
    if k*k != len(text):
        raise ValueError("Length of the text should be a perfect square.")

    if len(password) != k:
        raise ValueError(f"Length of the password should be {k}.")

    grid = generate_grid(k)
    encrypted_grid = encrypt_with_grid(grid, text)
    result = extract_by_password(encrypted_grid, password)

    return result

# Пример использования
text = "нужноподписатьновыйуказда"
password = "шифрр"
print(encrypt(text, password))
```

Пример работы программы

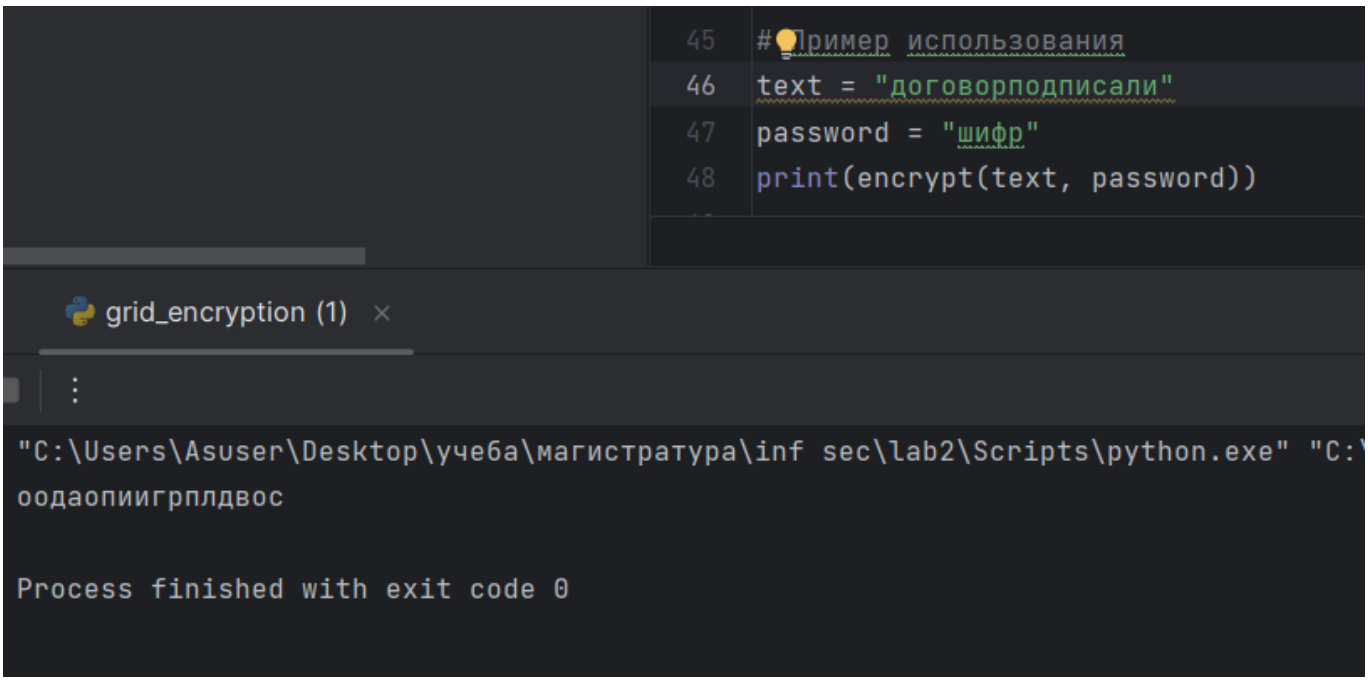


Таблица Виженера

В таблице записаны буквы русского алфавита. При переходе от одной строке к другой происходит циклический сдвиг на одну позицию. Пароль записывается с повторениями над буквами сообщения. В горизонтальном алфавите ищем букву нашего текста, а в вертикальном букву пароля и на их пересечении будет нужная буква.

Чтобы реализовать программу был написал след. код на python:

```python
1 usage
def vigenere_encrypt(text, key):
    alph = 'абвгдежзийклмнопрстуфхцчшщъыьэюя'
    encrypted_text = ''

    # размерность пароля = размерности текста
    key_expanded = ''
    while len(key_expanded) < len(text):
        key_expanded += key
    key_expanded = key_expanded[:len(text)]

    # сравниваем по таблице и получаем на пересечении букву
    for t, k in zip(text, key_expanded):
        if t in alph:
            new_pos = (alph.index(t) + alph.index(k)) % len(alph)
            encrypted_text += alph[new_pos]
        else:
            encrypted_text += t

    return encrypted_text


1 usage
def vigenere_decrypt(encrypted_text, key):
    alph = 'абвгдежзийклмнопрстуфхцчшщъыьэюя'
    decrypted_text = ''

```

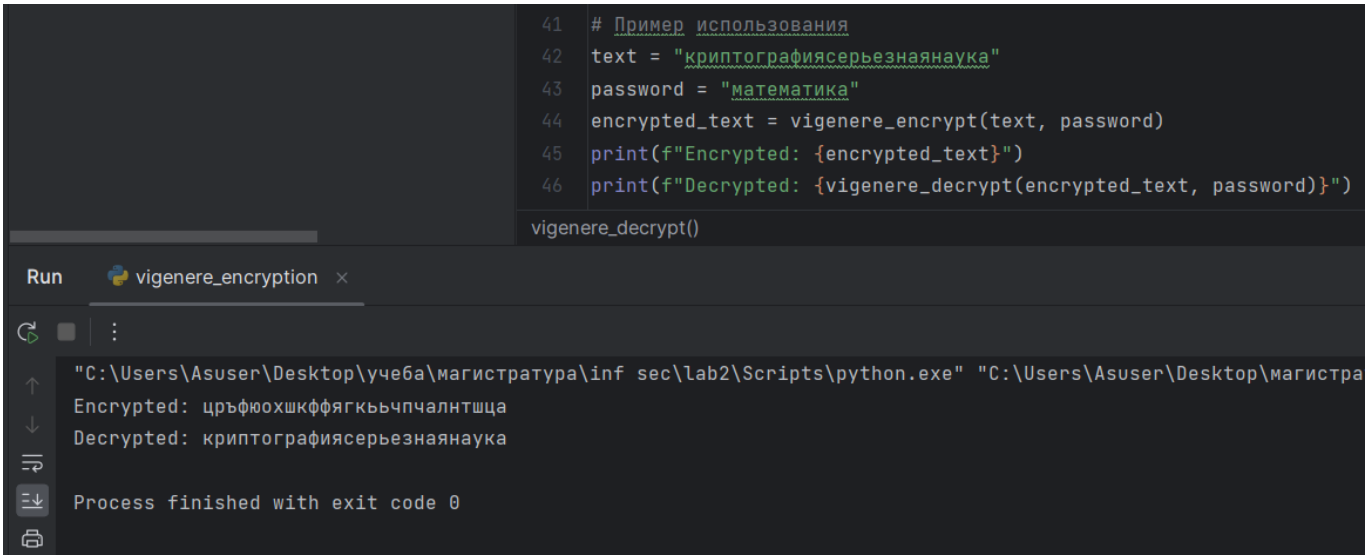
```
    key_expanded = ''
    while len(key_expanded) < len(encrypted_text):
        key_expanded += key
    key_expanded = key_expanded[:len(encrypted_text)]

    for e, k in zip(encrypted_text, key_expanded):
        if e in alph:
            new_pos = (alph.index(e) - alph.index(k)) % len(alph)
            decrypted_text += alph[new_pos]
        else:
            decrypted_text += e

    return decrypted_text


# Пример использования
text = "криптографиясерьезнаянаука"
password = "математика"
encrypted_text = vigenere_encrypt(text, password)
print(f"Encrypted: {encrypted_text}")
print(f"Decrypted: {vigenere_decrypt(encrypted_text, password)}")
```

Пример работы программы (как в методических материалах)



Выводы

В результате выполнения работы освоили на практике применение шифров перестановки.

Список литературы

1. Методические материалы курса